RCS 205
Operating Systems

Lecture 4:

System calls & Structure

Lecture 4:

- Scope:
  - System calls.
  - System programs.
  - Operating System structure.
- Goals:
  - Examine the different structural organizations for a modern operating system.
  - Understand the role played by system calls and system programs in the working of a modern operating system.

System calls

What is System calls ???

A **system call** is a request made by a program to the **operating system**. It allows an application to access functions and commands from the **operating system's** API.

System calls perform **system**-level operations, such as communicating with hardware devices and reading and writing files.

System calls:

- System calls provide the interface between a process and an operating system.
- These calls are generally available as *assembly-language instructions*.
- Some systems may allow system calls to be made directly from a higher-level language program.
- In such cases, system calls resemble predefined function and subroutine calls.

System calls…

- Some languages allow system calls to be made directly
  - Languages such as C and PERL, have been defined to replace assembly language for systems programming - their implementations include direct system call access.
  - Some Pascal systems also provide an ability to make system calls directly from a Pascal program to the O/S.
  - Most FORTRAN systems provide similar capabilities, often by a set of library routines.

System calls…

- Programs make heavy use of the operating system – however, most users don't see this level of detail since the run-time support for most programming languages provide a much simpler interface.

  For example:

  - Consider a simple program that reads data from one file and copy them to another file.
  - How many system calls are involved???

System calls…

- System calls occur in different ways, depending on the computer in use.
- Often, more information is required than simply the identity of the desired system call.
- The exact *type*, and *amount* of information vary according to the particular O/S and call.
  - For example:
    - To get input, we need to specify the file or device to use as the source, and the address and length of the memory buffer into which the input should be read.

System calls…

- During system calls, Information is passed in form of *parameters*.
- There are three general methods used to pass parameters to the O/s:
  1. The simplest method is to *pass the parameters in registers*.
  2. However, in some cases, the parameters may be more than the registers. In this case, they are *stored in a block or table in memory, and the address of the block is passed as a parameter in a register*.

System calls…

3. *Parameters can also be placed or pushed into a stack by the program, and popped off the stack by the operating system.*

- Some operating systems prefer the block or stack methods
  - The two have the advantage of not limiting the number, or length of parameters being passed.
  - Stack method is limited by the size of stack, and the stack variable types.

System calls – Types:

- System calls can be grouped into five major categories:
  - Process control.
  - File manipulation.
  - Device manipulation.
  - Information maintenance.
  - communication

System calls –Examples

Process control:

- End, abort.
- Load, execute.
- Create process, terminate process.
- Get process attributes, set process attributes.
- Wait for time.
- Wait event, signal event.
- Allocate and free memory.

System calls – Examples

File manipulation

- Create file, delete file.
- Open, close.
- Read, write, reposition.
- Get file attributes, set file attributes.

System calls –Examples

Device manipulation

- Request device, release device.
- Read, write, reposition.
- Get device attributes, set device attributes.
- Logically attach or detach devices.

System calls – Examples

Information maintenance:

- Get time or date, set time or date.
- Get system data, set system data.
- Get process, file, or device attributes.
- Set process, file, or device attributes.

System calls – Examples

Communications:

- Create, delete communication connection.
- Send, receive messages.
- Transfer status information.
- Attach or detach remote devices.

System Programs

System Programs:

- Another aspect of a modern operating system is the collection of system programs.
  - *They provide a more convenient environment for program development and execution.*
- Some system programs are simply user interfaces to system calls, whereas others are considerably more complex.
- The most important system program is the *command interpreter*.

System programs – Types:

- System programs can be divided into several categories:
  1. *File manipulation* – these programs generally manipulate files and directories.
  2. *Status information* – ask O/S for status information, format it, and have it printed to the terminal or other output device or file.
  3. *File modification* – several text editors may be available to create and modify the content of file stored on disk.

System programs – Types:

4. *Programming-language support* – compilers, assemblers and interpreters.
5. *Program loading and execution* – loading programs into memory for execution.
6. *Communications* – provide mechanisms for creating virtual connections among processes, users and different computer systems.
7. *Application programs* – to solve common problems, or to perform common operations.

Operating System Structure

Operating Systems Structure

- An O/S should be engineered carefully for proper functioning and easy modification.
  - Common approach is to structure system into small components rather than have a single large system.
- There are four major operating system designs structures:
  1. Monolithic Systems
  2. Layered Systems
  3. Virtual Machines
  4. Client-Server systems

1. Monolithic Systems

- This is the oldest and simplest sructure.
- Most common approach in early operating systems
- Also referred to as ‘big mess’.
- Frequently these O/S started small then grew big.

Monolithic systems…

- A monolithic O/S structure is characterized by:
  - The O/S written as a collection of procedures which can call any other procedures.
  - Each procedure has a well-defined interface, in term of the parameters and results.
  - Individual procedures are separately compiled to build the object code, then bound to a single object code using the *system linker*.
  - There is no encapsulation and information hiding - every procedure is visible to the other.

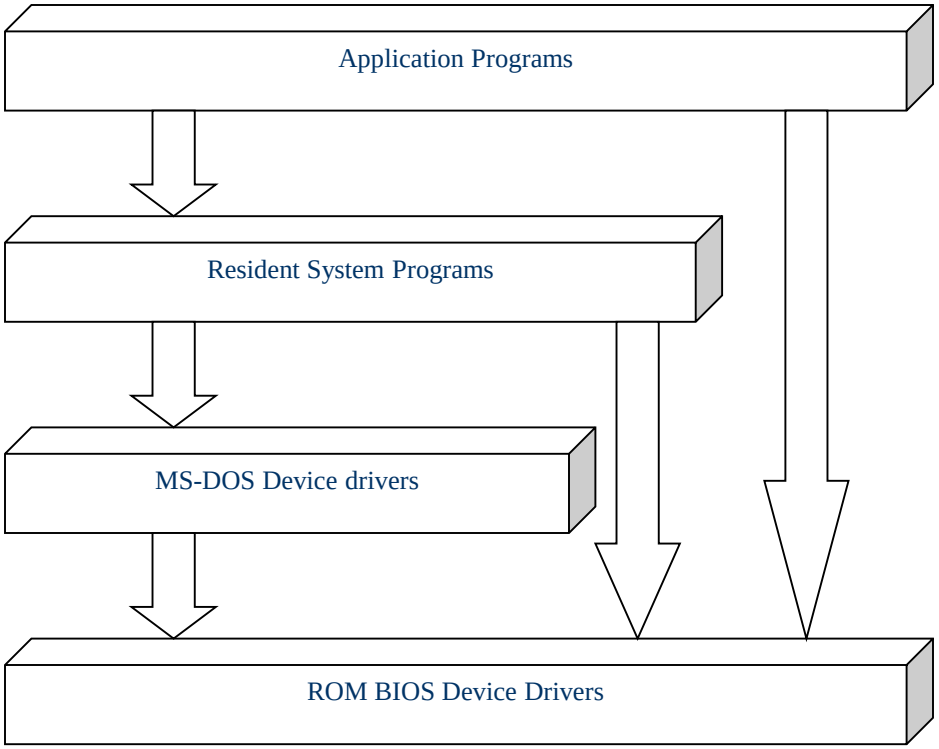
Monolithic Systems …

- System calls are requested by putting parameters in well defined places such as registers or the stack and then executing a *kernel* or *supervisor call*.
- This switches the machine from *user mode* to *kernel mode* and transfer control to the O/S.
  - In *kernel mode* (also known O/S mode), all instructions are allowed
  - *user mode* is used for user programs only. I/O and other system instructions are not allowed in this mode.
- In this model, user programs make system calls and when finished, control is given back to the user program

Monolithic Systems - example

- MS-DOS is an example of a monolithic system.
  - Designed and implemented by few big who didn't know it would become so popular and grow big.
  - Written to provide most functionality in in least space, due to limitation of h/w it ran on.
  - Not divided into modules carefully.
  - Interfaces and levels of functionality not well separated.

Monolithic Systems –Example…



MS DOS Layer Structure

2. Layered Systems

- In the layered structure, the O/S is broken down into a hierarchy of layers.
- Layers are selected such that each uses operations of the lower level layers only.
- This Simplifies debugging and system verification.
- The advantage of this approach is modularity.
- Each layer is constructed upon the layer below it.

Layered Systems - example

- The first layered system was THE (built at Technische Hogeschool Eindhoven) system built in Netherlands.
- This system had six layers:

Layer	Function
5	The operator
4	User Programs
3	I/O management
2	Operator-process communication
1	Memory and Drum management
0	Process allocation and multiprogramming

Layered Systems - example …

- **Layer 0** deals with allocation of the processor or context switching when timer interrupts occurred.
- **Layer 1** does memory management.
  - It allocates space for processes in main memory and pages (drum) for those that could not fit in memory.
  - Above layer 1 processes does not have to worry whether they were in main memory or the drum.
- **Layer 2** handles communication between each process and the operator console.

Layered Systems - example …

- **Layer 3** deals with the management the I/O devices and their buffers.
- **Layer 4** is where the user programs are found.
- **Layer 5** is where the system operator is located.

Layered systems…

- Disadvantages:
  - One disadvantage with this approach is that it *needs appropriate definition of various layers* as a layer can use only functions of lower layers.
    - For example, drives for backing storage must be at lower levels than those for memory management.
  - Another disadvantage is that they *tend to be less efficient than other approaches*.

Layered Systems …

- Example 2:
  - The MULTICS system presents a generalization of the layering concept.
    - Instead of layers, MULTICS was arranged as a series of concentric rings, with inner rings being more privileged than the outer ones.
    - A procedure in an outer ring had to make the equivalent of a system call when calling a procedure in the inner ring

3. Virtual Machines

- A time-sharing system provides multiprogramming and an extended machine with a more convenient interface that the bare hardware.
- Virtual machine system (e.g VM/370) separates these two functions.
  - Multiprogramming.
  - Provision of virtual machines.

Virtual machines…

- The core of the system, known as a *virtual machine monitor*, runs on bare hardware.
  - It does multiprogramming, providing not one but several virtual machines to the next layer up.
  - Each process is provided with a virtual copy of the underlying computer.
- Unlike other O/S, these virtual machines are not extended machines, but exact copies of the bare hardware.

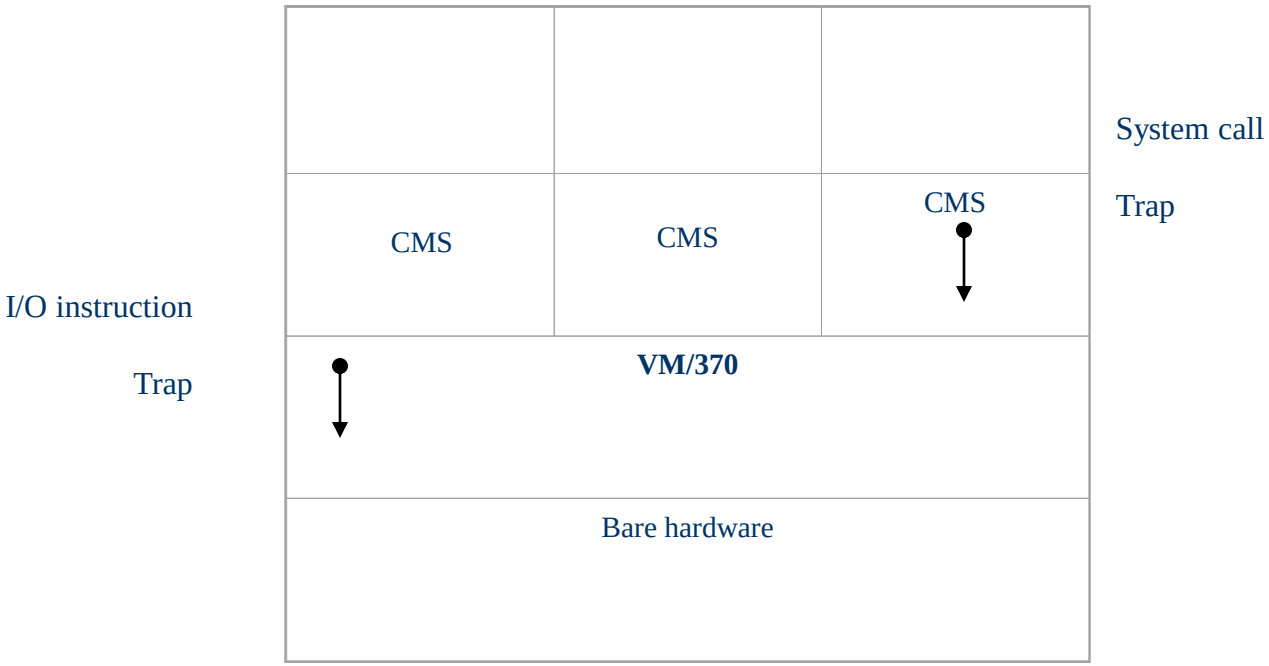
Virtual Machines …

- Each virtual machine can run any O/S that will run directly on the bare hardware.
- Different virtual machines can run different O/S.
- Some run a single-user, interactive system called CMS (Conversational Monitor System) for time-sharing users.

Virtual Machines …

- A CMS program system call is trapped to the O/S in its own virtual machine, not to VM/370.
- CMS then issues the normal hardware I/O instructions.
- I/O instructions are then trapped by VM/370.
- VM/370 then performs them as part of its simulation of the real hardware.
- This makes a complete separation of the functions of multiprogramming and providing an extended machine.
- Consequently, each of the O/S parts can be much simpler, more flexible, and easier to maintain.

Virtual Machines …



4. Client-Server Model

- Trend in modern O/S is remove as much function as possible, from the O/S, leaving a minimal *kernel*.
- The client-server approach implements most of O/S functions in user processes.
- Consequently, a bug in the server process may crash the server, but not the entire system.

Client-Server Model…

- Another advantage is its adaptability to use in distributed systems
  - If a client communicates with a server by sending it messages, the client need not know whether the message is handled locally or whether it was sent across the n/w to a server on a remote machine.

Client-Server Model…

- For example, to request a block of a file:
  - client process sends the request to the server process,
  - server process does the work and sends back the answer.
- In this model, the kernel handles the communication between clients and servers.

Client-server model…

- By splitting the O/S up into parts, each handling one facet of the system like file service, process service or memory service, each part becomes small and manageable.
- Further, servers run in user mode and not in kernel mode hence do not have direct access to the h/w